Django

PROYECTO FINAL
DESARROLLO DE APLICACIONES WEB

Oscar Bellón Caballero
2º DAW

ÍNDICE

INTRODUCCIÓN

Django es un web framework diseñado para *Python*, del que destaca la rapidez con la que puedes desarrollar gracias a él, su reutilización de código, y la cantidad de módulos de los que dispone.

METODOLOGÍA

Django se basa en el patrón Modelo-Vista-Controlador (MVC), pero reinterpretado. Según su parecer, la vista se define como "*lo que ves*", no "*cómo lo ves*", que sería la concepción clásica de la vista en el patrón MVC. Es por ello por lo su patrón deja el controlador de lado, lo pasa a llamar vista, y la vista clásica la sustituyen por el concepto *template* (plantilla). Quedando así lo que denominan Modelo-Vista-Template (MVT, o MTV).

MODELO

Se trata del lugar donde se guarda la información de la aplicación, es decir, la base de datos. Pero no se trata de una base de datos al uso, ya que utiliza el modelo *Object-Relational-Mapping* (ORM). Se trata de la transformación de un estilo de programación orientado a objetos en sentencias que entienda la base datos. Esto facilita la utilización de diferentes bases de datos, ya que el modelo ORM se encargará internamente de transformar nuestra programación orienta a objetos en las sentencias pertinentes.

```
class Post(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, default=1)
    title = models.CharField(max_length=120)
    slug = models.SlugField(unique=True)
    image = models.ImageField(upload_to=upload_location,
        null = True,
        blank=True,
        height_field="height_field",
        width_field="width_field")
    height_field = models.IntegerField(default=0)
    width_field = models.IntegerField(default=0)
    content = models.TextField()
    timestamp = models.DateTimeField(auto_now=False, auto_now_add=True)
    updated = models.DateTimeField(auto_now=True, auto_now_add=False)
    draft = models.BooleanField(default=False)
    publish = models.DateField(auto_now=False, auto_now_add=False)
```

Ejemplo de modelo en *Django*

```
queryset_list = Post.objects.filter(draft=False).filter(publish__lte=timezone.now())
```

Ejemplo de consulta ORM

VISTA

Como ya se ha dicho, se trata del controlador clásico. Se trata de la parte lógica de la aplicación. Se encargará de pedir datos al modelo, tratarlos, y pasarlos a la plantilla que corresponda.

```python
def post_list(request):
    queryset = Post.objects.all()
    context = {
        "queryset": queryset,
        "title": "First list"
    }
    return render(request, 'index.html', context)
```

Ejemplo de vista

TEMPLATE

En esta parte se visualizarían los datos que son pasados desde la vista. Se trata de código *HTML* en el que se pueden incluir las variables que han sido pasadas desde la vista (entre llaves), o bloques de código como condicionales o bucles.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Index</title>
  </head>
  <body>
    <h1> Hello from templates!</h1>
    <h2>Your context is: {{title}}</h2>
    <ul>
    {% for post in queryset %}
      <li>{{ post.title }}</li>
    {% endfor %}
    </ul>
  </body>
</html>
```

Ejemplo de plantilla

FLUJO

El flujo de trabajo en *Django* es bastante simple, lo que facilita las cosas cuando se quieren añadir funcionalidades nuevas en la aplicación.

Primero, la *url* introducida en el navegador se pasa al archivo *urls.py*, aquí, y mediante una expresión regular, se decide a qué vista se le pasa el control.

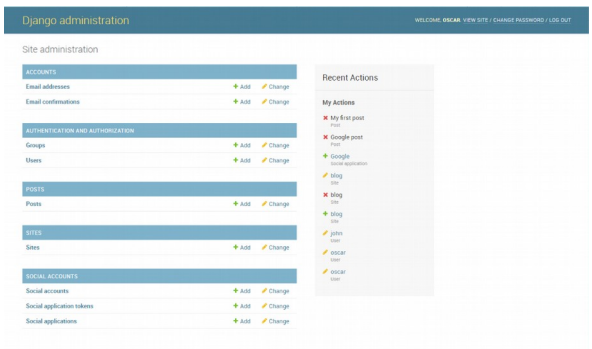
```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.post_list, name='list'),
    url(r'^myposts$', views.my_posts, name='myposts'),
    url(r'^create/$', views.post_create),
    url(r'^(?P<slug>[\w-]+)/$', views.post_detail, name='detail'),
    url(r'^(?P<slug>[\w-]+)/edit/$', views.post_update, name='update'),
    url(r'^(?P<slug>[\w-]+)/delete/$', views.post_delete),

]
```

Ejemplo de archivo *urls.py*

Segundo, el control llega a la vista decidida, que se encontrará en el archivo *views.py.* En este archivo se puede decidir o bien trabajar con las vistas como funciones, o bien como clases: las vistas como funciones son mucho más sencillas de realizar, mientras que las clases aportan robustez y mayor funcionalidad. Cuando la vista llega a su fin, redirige el flujo hacia una plantilla.
Tercero, el control llega a la plantilla elegida, y desde aquí se muestran los datos que se han pasado desde la vista.
El círculo se cerraría cuando el usuario interactuase con la plantilla y redirigiese el flujo hacia otra *url*, que sería analizada por el archivo *urls.py*, y volvería a empezar el proceso.

ADMINISTRADOR

Se trata de una herramienta totalmente editable desde el propio código fuente, que facilita el manejo de la información que se almacena en nuestra aplicación. Desde los modelos, hasta los usuarios registrados, o las aplicaciones sociales que se tengan.



Administrador de *Django*

CONCLUSIÓN

Se ha demostrado que *Django* permite el desarrollo de manera eficaz, con un flujo de trabajo muy estricto, pero simple y muy fácil de comprender. Por otra parte, la gran cantidad de módulos que vienen integrados con él facilita su uso, evitando los problemas de importaciones que se pueden sufrir al utilizar otros *frameworks.* Su comunidad es otro de los puntos fuertes, con usuarios muy activos, y una documentación excelente.
Es por ello por lo que creo que *Django* es una gran apuesta para proyectos tanto medianos como grandes, por su escalabilidad, y focalización en el desarrollo del código.

BIBLIOGRAFÍA

Página oficial de *Django* - https://www.djangoproject.com
Tutorial 1 - http://tutorial.djangogirls.org/en
Tutorial 2 - http://www.marinamele.com/taskbuster-django-tutorial
Tutorial 3 (serie de vídeos) - https://www.codingforentrepreneurs.com/projects/try-django-19
Lista interminable de recursos sobre *Django* - http://awesome-django.com